

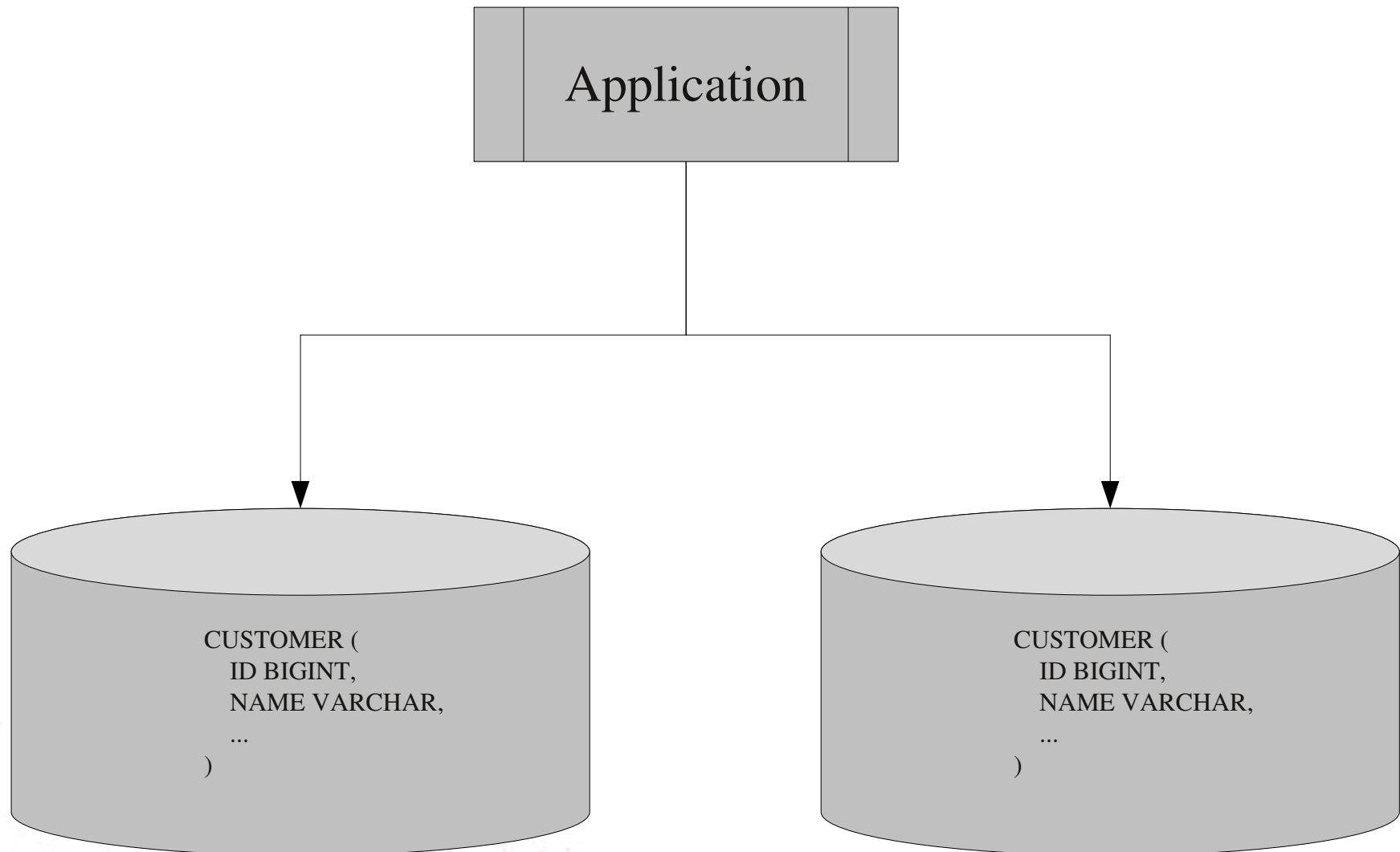
# Multi-tenancy in Hibernate

Steve Ebersole  
Hibernate Project Lead

# Agenda

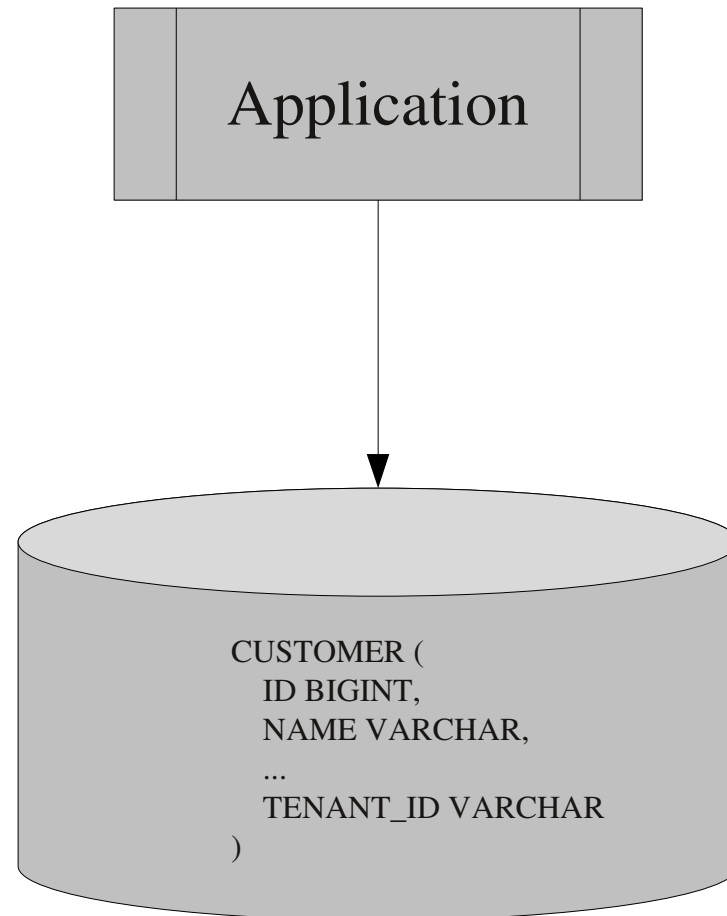
- What is multi-tenancy
- Current options in Hibernate
- Support in Hibernate 4
- Q & A

# Separate Schema



Multi-tenancy in Hibernate | Steve Ebersole

# Discriminator



# Current Options - Discriminator

- Hibernate filters
- This approach can still leverage 2<sup>nd</sup> level cache
- Application must be aware of tenant id
  - Entity must expose tenant-id attribute
  - Application must set “tenant id” on creation

# Discriminator - Entity

```
@Entity
@FilterDef( name="tenantFilter", parameters=@ParamDef( name="tenantId", type="string" ) )
@Filters(
    @Filter( name="tenantFilter", condition="tenant_id = :tenantId" )
)
public class Customer {
    @Id
    private Long id;
    @Column( name="tenant_id", nullable=false, updateable=false )
    private String tenantId;
    ...
}
```

# Discriminator - Usage

```
public void demonstrateUsageInDiscriminatorScenario() {
    // One thing that must be done on all sessions is to enable the tenant-based Hibernate filter
    // we defined before on the entity. This is usually best handled by some form of "request
    // interceptor" (HttpServletRequestFilter, etc) to make sure it is done uniformly

    // Creating an entity
    Session session = openSession();
    session.enableFilter( "tenantFilter" ).setParameter( "tenantId", "some-tenant-identifier" );
    session.beginTransaction();
    Customer customer = new Customer();
    ...
    customer.setTenantId( "some-tenant-identifier" );
    session.persist( customer );
    session.getTransaction().commit();
    session.close();

    // Querying Customers
    session = openSession();
    session.enableFilter( "tenantFilter" ).setParameter( "tenantId", "some-tenant-identifier" );
    session.beginTransaction();
    Customer customer = (Customer) session.createQuery( "from Customer" ).uniqueResult();
    session.getTransaction().commit();
    session.close();
}
```

# Current Options – Separate Schema

- Custom ConnectionProvider to route calls to the correct JDBC connection
- Not safe with 2<sup>nd</sup> level cache
- Application not aware of tenant-id



# Separate Schema - Entity

```
@Entity
public class Customer {
    @Id
    private Long id;
    ...
}
```

# Separate Schema - ConnectionProvider

```
public class MyTenantAwareConnectionProvider implements ConnectionProvider {
    public static final String BASE_JNDI_NAME = "java:/comp/env/jdbc/";

    public Connection getConnection() throws SQLException {
        final String tenantId = TenantContext.getTenantId()
        final String tenantDataSourceName = BASE_JNDI_NAME + tenantId;
        DataSource tenantDataSource = JndiHelper.lookupDataSource( tenantDataSourceName );
        return tenantDataSource.getConnection();
    }

    public void closeConnection(Connection conn) throws SQLException {
        conn.close();
    }

    public boolean supportsAggressiveRelease() {
        // so long as the tenant identifier remains available in TenantContext throughout, we can
        // reacquire later
        return true;
    }

    public void configure(Properties props) {
        // currently nothing to do here
    }

    public close() {
        // currently nothing to do here
    }
}
```

# Separate Schema - Usage

```
public void demonstrateUsageInSeparateSchemaScenario() {
    // The ConnectionProvider we saw earlier is registered with the SessionFactory as the means
    // for Hibernate to acquire Connections as needed for the Session. Here we must push the
    // tenant-id to the TenantContext so it is available to the ConnectionProvider. This is usually
    // best handled by some form of "request interceptor" (HttpServletRequestFilter, etc) to make
    // sure it is done uniformly

    TenantContext.setTenantId( "some-tenant-identifier" );

    // Creating an entity
    Session session = openSession();
    session.beginTransaction();
    Customer customer = new Customer();
    ...
    session.persist( customer );
    session.getTransaction().commit();
    session.close();

    // Querying Customers
    session = openSession();
    session.beginTransaction();
    Customer customer = (Customer) session.createQuery( "from Customer" ).uniqueResult();
    session.getTransaction().commit();
    session.close();
}
```

# Support in Hibernate 4

- Exact API still under discussion[1]
- Public API options:
  - `Session.setTenantId(String tenantId)`
  - Passed as part of opening a Session
- Transparently handled by Hibernate

[1] <http://opensource.atlassian.com/projects/hibernate/browse/HHH-5697>